Universal Serial Bus
Device Class Definition
for Audio Devices

0.8a Draft Revision
February 5, 1996

Scope of this Revision

The 0.8a release candidate of this definition is intended for industry review.

Contributors

Geert Knapen
Philips ITCL
Interleuvenlaan 74-76
B-3001 Leuven-Heverlee BELGIUM
Phone: +32 16 390 734
Fax: +32 16 390 600
e-mail: gkn@belvits0.serigate.philips.nl

Revision History

Revision	Date	Filename	Author	Description
0.1	Aug 7, 95	Audio01.doc	Geert Knapen	Initial version
0.2	Aug 28, 95	Audio01.doc	Geert Knapen	Corrected typos. Attributes field from 8 to 16 bits. Auxiliary channel definition Important issues added.
0.3	Oct 9, 95	Audio03.doc	Geert Knapen	Intermediate version
0.4	Nov 29, 95	Audio04.doc	Geert Knapen	Change to Audio Function and Interface Property requests Synch issues updated Subclass divisions changed
0.6	Dec 19, 95	Audio06.doc	Geert Knapen	Listed remarks from last f2f Dec 7-8
0.8	Dec 12, 95	Audio08.doc	Geert Knapen	Incorporated changes, discussed at f2f Dec 6 95
0.8a	Jan 20, 96	Audio08a.doc	Geert Knapen	Incorporated changes discussed at f2f Jan 18 95. Feedforward/feedback endpoint is now called sync endpoint.
	Feb 5, 96	usb_au8a.doc		Edited version of Audio08a.doc

USB Device Class Definition for Audio Devices

INTELLECTUAL PROPERTY DISCLAIMER

Please send comments via electronic mail to usbdevice@fsp008.fm.intel.com

Table of Contents

List of Tables

1. Introduction

1.1 Scope

The Audio Device Class Definition applies to all devices or functions embedded in composite devices that are used to manipulate audio, voice, and sound-related functionality. This includes both audio data (analog and digital) and the functionality that is used to directly control the audio environment, such as volume and tone control. The Audio Device Class does not include functionality to operate transport mechanisms that are related to the reproduction of audio data, such as tape transport mechanisms or CD-ROM drive control. Although the handling of MIDI data streams over the USB is related to audio, that topic is not covered in this document.

1.2 Purpose

The purpose of this document is to describe the minimum capabilities and characteristics an audio device must support in order to comply with the USB. This document also provides recommendations for optional features.

1.3 Related Documents

- *Universal Serial Bus Specification*, 1.0 final draft revision (also referred to as the *USB Specification*). In particular, see Chapter 9, “USB Device Framework.”
- IEC 958 International Standard: *Digital Audio Interface and Annexes*
- ANSI S1.11-1986 standard

1.4 Terms and Abbreviations

This section defines terms used throughout this document. For additional terms that pertain to the Universal Serial Bus, see Chapter 2, “Terms and Abbreviations,” in the *USB Specification*.

Audio data stream	A transport medium that can carry a single source of audio information.
Audio data stream cluster	A group of audio data streams that carry tightly related synchronous audio information. A normal stereo audio stream is a typical example of a two-channel audio data stream cluster.

2. Management Overview

The USB is very well suited for transport of audio (voice and sound). PC-based voice telephony is one of the major drivers of USB technology. Also, the USB has more than enough bandwidth for sound, even high quality audio. Many applications related to voice telephony, audio playback, and recording can take advantage of the USB.

In principle, a versatile bus specification like the USB provides many ways to propagate and control digital audio. For the industry, however, it is very important that audio transport mechanisms be well defined and standardized on the USB. Only in this way can interoperability be guaranteed among the many possible audio devices on the USB. Standardized audio transport mechanisms also help to keep software drivers as generic as possible. The Audio Device Class described in this document satisfies those needs. It is written and revised by experts in the audio field. Other device classes that deal with audio in some way should refer to this document for their audio interface specification.

A key issue in audio is synchronization of the data streams. Indeed, the smallest artifacts are easily detected by the human ear. Therefore, a robust synchronization scheme on isochronous transfers has been developed and incorporated in the *USB Specification.* The Audio Device Class definition adheres to this synchronization scheme to transport audio data reliably over the bus.

This document contains all necessary information for a designer to build a USB-compliant device that incorporates audio functionality. It specifies the standard and class-specific descriptors that must be present in each USB audio function. It further explains the use of class-specific requests that allow for full audio function control. A number of predefined subclass/protocol combinations are listed and fully documented. Each combination defines a standard way of transporting audio over USB. However, provisions have been made so that vendor-specific audio formats and compression schemes can also be handled.

3. Functional Characteristics

In many cases, audio functionality does not exist as a standalone device. It is one capability that, together with other functions, constitutes a “composite” device. A perfect example of this is a CD-ROM player, which can incorporate video, audio, data storage, and transport control. The audio function is thus located at the interface level in the device class hierarchy, and consists of a number of related pipes that together implement the audio interface.

Audio functions are addressed through their audio interfaces. Each audio function has a single audio interface. A device can have multiple audio interfaces active at the same time. These interfaces are used to control multiple independent audio functions located in the same device.

3.1 Audio Interface Class

The Audio Interface class groups all functions that can interact with USB-compliant audio data streams. All functions that convert between analog and digital audio domains can be part of this class. Also those functions that transform USB-compliant audio data streams into other USB-compliant audio data streams can be part of this class. Even analog audio functions that are controlled through USB belong to this class.

In fact, for an audio function to be part of this class, the only required property it must support is the master mute property. No further interaction with the function is mandatory, although most functions in the audio interface class will support one or more optional features like consuming or producing one or more isochronous audio data streams, volume or level control, multiple sampling frequency support etc.

The Audio Interface class code is assigned by the USB. For details, see Appendix 0 A.1 Audio Interface Class Code.

3.2 Audio Interface Subclass and Protocol

The Audio Interface class is divided into subclasses which are further qualified by the interface protocol code. All audio functions are part of a certain subclass/protocol. The subclass/protocol provides pertinent information about the audio function’s capabilities, the data formats it uses, the compression schemes it supports, and so forth. A number of predefined subclasses are general, and are fully defined in this document. These functions have a subclass code assigned in the range 0x01 to 0x7F. The predefined subclass/protocol codes can be found in Appendix A. Other subclasses are vendor-specific and are not covered in this document. Their assigned subclass codes can range from 0x80 to 0xFF.[1]

3.3 Audio Synchronization Types

Each isochronous audio endpoint used in an audio interface belongs to a synchronization type as defined in Chapter 5 of the *USB Specification*. The following sections briefly describe the possible synchronization types.

3.3.1 Asynchronous

Asynchronous isochronous audio endpoints produce or consume data at a rate that is locked either to a clock external to the USB or to a free-running internal clock. These endpoints cannot be synchronized to a start of frame (SOF) or to any other clock in the USB domain.

---

[1] This range differs from the *USB Specification*, which states that only subclass code 0xFF is reserved for vendor-specific subclasses.

3.3.2 Synchronous

The clock system of synchronous isochronous audio endpoint can be controlled externally through SOF synchronization. Such an endpoint must do one of the following:

- Slave its sample clock to the 1 ms SOF tick.
- Control the rate of USB SOF generation so that its data rate becomes automatically locked to SOF.

3.3.3 Adaptive

Adaptive isochronous audio endpoints are able to source or sink data at any rate within their operating range. This implies that these endpoints must run an SRC process internally that allows them to match their natural data rate to the data rate that is imposed at their interface.

3.4 Audio Data Stream Packets

Audio data streams that are inherently continuous must be packetized when sent over the USB. The quality of the packetizing algorithm directly influences the amount of effort needed to reconstruct a reliable sample clock at the receiving side. The goal must be to keep the instantaneous number of samples per frame (n_i) as close as possible to the average number of samples per frame, (n_{av}).

If the sampling rate is a constant, the allowable variation on n_i is limited to one sample, that is, $\Delta n_i = 1$. This implies that all packets must either contain INT (n_{av}) (small packet) or INT (n_{av}) + 1 (large packet) samples. For all *i*:

$$ni = INT(nav) \mid INT(nav) + 1$$

If the sampling rate can be varied (in order to implement pitch control), the allowable pitch shift is 1kHz/ms, that is, the allowable variation on n_i is limited to one sample per frame. For all *i*:

$$n_{i+1} = n_i \pm 1$$

Pitch control is restricted to adaptive endpoints only. Audio functions that support pitch control on an endpoint are required to report this in the class-specific endpoint descriptor. In addition, a **PitchControl** request is required to enable or disable the pitch control property.

3.5 Data Interleaving

Each packet sent between host and an isochronous audio data endpoint contains parts of all audio data streams that belong to the cluster associated with the endpoint. Data is packaged in an interleaved format in which the samples at time *x* of all subsequent channels of the cluster are sent first, followed by the samples at time *x*+1 and so on, until all samples are transmitted. Each packet always starts with the same channel and the channel order is respected throughout the entire transmission.

3.6 Interchannel Synchronization

An important issue when dealing with audio, and 3-D audio in particular, is the phase relationship between different physical audio channels. Indeed, the virtual spatial position of an audio source is directly related to and influenced by the phase differences that are applied to the different physical audio channels used to reproduce the audio source. Therefore, it is imperative that USB audio functions respect the phase relationship among all related audio channels. However, the responsibility for maintaining the phase relation is shared between the USB host software and hardware and all of the audio peripheral devices or functions.

In order to provide a manageable phase model to the host, an audio function is required to report its internal delay. This delay is expressed in number of frames (ms) and is due to the fact that the audio function must buffer at least one frame's worth of samples to effectively remove packet jitter within a frame. Furthermore, some audio functions will introduce extra delay because they need time to correctly interpret and process the audio data streams (for example compression and decompression). However, it is required that an audio function introduce only an integer number of frames of delay. In case of an audio source function, this implies that the audio function must guarantee that the first sample it fully acquires after SOF*n* (start of frame *n*) is the first sample of the packet it sends over USB during frame ($n+\delta$), where δ is the audio function's internal delay expressed in ms. For an audio sink function, the same rule applies. The first sample in the packet, received over USB during frame *n*, must be the first sample that is fully reproduced during frame ($n+\delta$).

By following these rules, phase jitter is limited to ±1 audio sample. It is up to the host software to synchronize the different audio streams by scheduling the correct packets at the correct moment, taking into account the internal delays of all audio functions involved.

3.7 Operational Model

The following sections describe the types of endpoints that are used in an audio interface for audio function control and for audio data stream transfer.

3.7.1 Audio Interface Description

As stated earlier, audio functionality is located at the interface level in the device class hierarchy and consists of a number of related endpoints that together implement the interface of a particular audio function. These endpoints can be:

- A control endpoint for manipulating settings and retrieving the state of the audio function. This endpoint is mandatory, and the default endpoint 0 is used for this purpose.
- An interrupt endpoint for status return. This endpoint is also mandatory, and the shared interrupt endpoint is used for this purpose.
- One or more isochronous endpoints, in some cases with associated sync endpoints, for audio data transfer. These endpoints are optional.

A device can support multiple configurations. Within each configuration can be multiple interfaces, each possibly having alternate settings. Several different and independent audio interfaces can exist in the same device.

As an example, consider a PC monitor equipped with a built-in stereo speaker system. Such a device could be configured to have one interface dealing with configuration and control of the monitor part of the device, while a second interface deals with its audio aspects.

The audio interface could be configured to operate in mono mode (alternate setting 0) in which only a single channel data stream is sent to the audio function. This audio stream could then be reproduced on both speakers. From an interface point of view, such a setup requires one isochronous endpoint to receive the mono audio data stream, in addition to the mandatory control endpoint and interrupt endpoint.

The same system could be used to play back stereo audio. In this case, the stereo interface must be selected (alternate setting 1). This interface also consists of a control endpoint, an interrupt endpoint, and a single isochronous endpoint, now receiving a data stream that interleaves left and right channel samples.

If the above audio interface were an asynchronous sink, one extra isochronous sync endpoint would also be necessary.

The following sections describe audio-related endpoints in more detail.

3.7.1.1 Isochronous Audio Data Stream Endpoint

In general, the data streams that are handled by an isochronous audio data endpoint do not necessarily map directly to the physical channels that exist within the audio function. As an example, consider a “stereo” audio data stream that contains audio data encoded in Dolby Prologic™ format. Although there are only two data streams (left and right), these two channels carry information for four physical channels (left, right, center and surround). Other examples include cases in which multiple physical audio channels are compressed into a single data stream. The format of such a data stream can be entirely different from the native format of the physical channels (for example, 256 kbits/s MPEG1 stereo audio as opposed to 176.4 kBytes/s 16 bit stereo 44.1 kHz audio). Therefore, in order to correctly describe data transfer at the endpoint level, the notion of physical channel is replaced by the notion of audio data stream.

As a consequence, requests to control physical properties that exist within an audio function, such as volume or mute, cannot be sent to an endpoint because it operates on audio data streams and is unaware of the number of physical channels it eventually serves. Instead, these requests must be directed to the audio function’s interface via control endpoint 0.[2]

As already mentioned, a single audio interface can have zero or more isochronous endpoints. If multiple synchronous data streams must be communicated between host and audio function, these streams, which share all characteristics typical for the endpoint such as synchronization type, sampling frequency, bits/sample, and so forth, can be clustered into one audio stream cluster by interleaving the individual audio data, and the result can be directed to a single endpoint. Furthermore, a single sync endpoint, if needed, can service the entire cluster. In this way, a minimum number of endpoints is consumed to transport related data streams.

If an audio function needs more than one cluster to operate, each cluster is directed to a separate endpoint. An example of such an audio function is a digital mixer with two stereo input channels (four channels in all) and one stereo output channel. In addition to the control endpoint and the status interrupt endpoint, this audio function’s interface could consist of two isochronous endpoints, one for receiving the four interleaved input channels if they were all synchronous, and the other for returning the two interleaved output channels. If the two stereo input channels were asynchronous, three isochronous endpoints would be needed.

Isochronous audio data stream endpoints are optional. If the audio interface controls only analog audio functions, there is no need for isochronous endpoints.

3.7.1.2 Isochronous Sync Endpoint

For adaptive audio source endpoints and asynchronous audio sink endpoints, an explicit sync mechanism is needed in order to maintain synchronization during transfers. For details about synchronization, see Chapter 5, “USB Data Flow Model,” in the *USB Specification*.

The information carried over the sync path consists of a 4-byte data packet. The first three bytes contain the F_f value in a 10.14 format as described in section 5.10.4.2 of the *USB Specification*. The fourth byte is a CRC value that protects the F_f value. F_f represents the average number of samples the endpoint must produce or consume per frame in order to exactly match the desired sampling frequency F_s.

---

[2] In many common cases, there is a one-to-one mapping between audio data streams and physical channels (for example, 16-bit stereo PWM audio data, sent to a two-stream endpoint). Even in this case, channel property control is established at the audio interface level and not at the endpoint level.

A new F_f value is available every 2^(10 – P) ms (frames) where P can range from 1 to 9 inclusive. The sample clock F_s is always derived from a master clock F_m in the device. P is related to the ratio between those clocks through the following relationship:

$$F_m = F_s * 2^{(P - 1)}$$

In worst case conditions, only F_s is available and $F_m = F_s$, giving P = 1 since one can always use phase information to resolve the estimation of F_s within half a clock cycle.

An adaptive audio source IN endpoint is accompanied by an associated isochronous sync OUT endpoint that carries F_f. (An asynchronous audio sink OUT endpoint is accompanied by an associated isochronous sync IN endpoint). The link between the two endpoints is established through the **bEndpointLink** field in the class specific endpoint descriptor of both endpoints. These fields contain the endpoint number of the other associated endpoint. Since an isochronous pipe is serviced on a per frame basis, the same value for F_f is sent over the sync endpoint during 2^(10 – P) frames.

The isochronous sync endpoint reports its P value in the **bUpdateCode** field of the endpoint descriptor. It can be used by the driver to calculate the update rate for F_f.

3.7.1.3 Control Endpoint

The audio interface class uses endpoint 0 (the default pipe) as the standard way to control the audio function through the use of class-specific requests. The format and contents of these requests are detailed further in this document.

In order to be able to manipulate the physical properties of an audio function, its functionality must be divided into addressable entities. Such a generic entity is called an Audio Control Block (ACB). It is the basic building block that has enough functionality incorporated so that most audio functions can be fully described by connecting together a number of these ACBs according to a certain topology.

A maximally equipped ACB supports all the general properties that are defined in this document. These general properties are:

- Mute
- Volume
- Bass
- Mid
- Treble
- Graphic Equalizer
- Emphasis[3]

The class-specific interface descriptor contains an array of Audio Control Block descriptors (ACBD), each specifying the properties that every ACB supports. The zero-based index in the array serves as a unique identifier for each ACB in the audio function. This identifier is passed along with every **Set/Get Property** request.

An audio function is required to support at least ACB[0], which is used to control the master general properties of the audio function (master volume, master tone control, and so forth). All master general properties except for the master mute property are optional. Every audio function must implement the master mute property.

---

[3] Explain the term emphasis here

The Audio Control Block descriptor contains a bitmap field that indicates the general properties that the associated ACB supports. In addition, there is an extension field that can be used by vendors to indicate that the ACB has extended (vendor-specific) capabilities. The code in this field directly refers to the nature of these extended capabilities. It is the vendor's responsibility to provide all of the necessary information to operate the ACB to its full extent. This could be accomplished through a vendor-specific interface descriptor. The ACB also contains an index to an optional string descriptor that describes the ACB.

Note that the class-specific interface descriptor provides no information concerning the topology of the audio function, that is, how the ACBs are connected—it merely provides a means to address all programmable entities in the audio function. How the ACBs are connected is implicitly described by the subclass/protocol, since these values fully characterize the inner workings of the audio function.

By grouping common basic functionality into a generic control block, a large portion of conceivable audio functions can be described and controlled in a generic fashion. The extension field allows further control of functionality not covered by this document.

3.7.1.4 Status Interrupt Endpoint

This paragraph must be adjusted when information on shared interrupt endpoints becomes available

A USB audio interface must support an interrupt endpoint in order to inform the host about the status of the audio function. Not only changes of state are reported through the interrupt endpoint, but the status information itself is contained in the interrupt data. The interrupt data is a 1-byte bitmap. The following table specifies the format of the status byte.

Table 3-1: Status Byte Format

Bit	BitName	Description
0	sError	If set, an error occurred. More detailed information can be obtained using the **Get Error** request.
1	sUnlocked	If set, at least one endpoint is out of lock. If clear, indicates that all isochronous endpoints of this interface are reconstructing a reliable sample clock. More detailed information can be obtained using the **Get Status** request.
2	sOverflow	If set, a conversion overflow occurred since the last interrupt. More detailed information can be obtained using the **Get Status** request.
3	sUserInput	If set, a user-initiated event occurred (for example, manual volume change).
4..6	Reserved	
7	sChangeOfState	If set, one of the other status bits has changed since the last status byte.

3.7.2 Audio Data Format

The format used to transport audio data over the USB is entirely determined by the audio interface subclass/protocol code. Therefore, each defined subclass/protocol must document in detail the layout of its interface and the audio data format it uses. For details about the predefined subclass/protocol data formats, see section 6, "Subclasses and Protocols." Vendor-specific subclass/protocols must be fully documented by the manufacturer.

4. Descriptors

The following sections describe the standard and class-specific USB descriptors for the Audio Interface Class.

4.1 Device Descriptor

Since audio functionality is always considered to reside at the interface level, there is no specific audio device descriptor. The device descriptor is thus dictated by other device class definitions. Exceptions to this rule are those devices that incorporate only audio functions. Such devices belong to the Audio Device Class and the device descriptor indicates that the device is audio only by specifying the Audio Interface Class code in its **bDeviceClass** field. The **bDeviceSubClass** and **bDeviceProtocol** fields must be set to 0. All other fields of the device descriptor must comply with the definitions in section 9.7.1 of the *USB Specification*. There is no class-specific device descriptor.

4.2 Configuration Descriptor

As for the device descriptor, an audio configuration descriptor is applicable only in the case of audio-only devices. It is identical to the standard configuration descriptor defined in section 9.7.2 of the USB Specification. There is no class-specific configuration descriptor.

4.3 Interface Descriptors

The interface descriptors contain all relevant information to fully characterize the corresponding audio function. The standard interface descriptor characterizes the interface itself, whereas the class-specific interface descriptor provides pertinent information concerning the internals of the audio function. It specifies the number of ACBs supported by the audio function and lists the capabilities of each ACB.

4.3.1 Standard Interface Descriptor

The standard audio interface descriptor is almost identical to the standard interface descriptor defined in section 9.7.3 of the *USB Specification*, except for the meaning of the subclass/protocol codes.

Table 4-1: Standard Interface Descriptor

Offset	Field	Size	Value	Description
0	bLength	1	Number	Size of this descriptor in bytes: 9
1	bDescriptorType	1	Constant	INTERFACE descriptor type
2	bInterfaceNumber	1	Number	Number of interface. A zero-based value identifying the index in the array of concurrent interfaces supported by this configuration.
3	bAlternateSetting	1	Number	Value used to select an alternate setting for the interface identified in the prior field.
4	bNumEndpoints	1	Number	Number of endpoints used by this interface (excluding endpoint 0). This number must be at least 1 since the status interrupt endpoint is mandatory.
5	bInterfaceClass	1	Class	C_AUDIO Audio Interface Class code (assigned by the USB). See Appendix A.
6	bInterfaceSubClass	1	Subclass	Subclass code. Subclass codes in the range 0x01..0x7F are predefined and assigned by USB. See Appendix A. Subclass codes in the range 0x80..0xFF are vendor-specific.
7	bInterfaceProtocol	1	Protocol	Further qualification of the subclass code. See Appendix A.
8	iInterface	1	Index	Index of a string descriptor that describes this interface.

4.3.2 Class-Specific Interface Descriptor

The length of the class-specific interface descriptor depends on the number of ACBs that are supported by the audio function described by this interface. The class-specific interface descriptor consists of a header followed by a list of audio control block descriptors (ACBDs). The total length of the class specific interface descriptor is then given by:

TotalLength = 3 + bNumACBs * ACBDLength

ACB’s are numbered from 0 to bNumACBs – 1 and ACBDLength = 3.

Table 4-2: Class-Specific Interface Descriptor Header

Offset	Field	Size	Value	Description
0	bLength	1	Number	Size of this descriptor in bytes (TotalLength as calculated above).
1	bDescriptorType	1	Constant	INTERFACE descriptor type.
2	bNumACBs	1	Number	Number of ACBs supported by this interface.

The following Audio Control Block descriptor (ACBD) is appended for each ACB in the function.

Table 4-3: Audio Control Block Descriptor

Offset	Field	Size	Value	Description
3+(i-1)*3	bmGenACBProps[i]	1	Bitmap	A bit set to 1 indicates that the mentioned property is supported for ACB[i]: Bit 0 Mute property Bit 1 Volume property Bit 2 Bass property Bit 3 Mid property Bit 4 Treble property Bit 5 Graphic Equalizer property Bit 6 Emphasis property Bit 7 System-exclusive properties
3+(i-1)*3 +1	bSysExCode	1	Number	Vendor-specific code indicating extended ACB capabilities not covered by this document.
3+(i-1)*3 +2	iChannel[i]	1	Index	Index of a string descriptor describing ACB[i] channel i.

4.4 Endpoint Descriptors

The following sections describe all possible endpoint-related descriptors.

4.4.1 Control Endpoint Descriptor

Since endpoint 0 is used as the control endpoint, there is no dedicated standard or class-specific control endpoint descriptor.

4.4.2 Status Interrupt Endpoint Descriptor

The status interrupt endpoint descriptor is identical to the standard endpoint descriptor defined in section 9.7.4 of the *USB Specification*. Its fields are set to reflect the interrupt type of the endpoint. There is no class-specific status interrupt endpoint descriptor. This endpoint is mandatory.

Table 4-4: Status Interrupt Endpoint Descriptor

Offset	Field	Size	Value	Description
0	bLength	1	Number	Size of this descriptor in bytes: 7
1	bDescriptorType	1	Constant	ENDPOINT descriptor type
2	bEndpointAddress	1	Endpoint	The address of the endpoint on the USB device described by this descriptor. The address is encoded as follows: Bit 0..3: The endpoint number, determined by the designer. Bit 4..6: Reserved, reset to zero Bit 7: Direction. 1 = IN endpoint
3	bmAttributes	1	Bit Map	Bit 0..1: Transfer type 11 = Interrupt All other bits are reserved.
4	wMaxPacketSize	2	Number	Maximum packet size this endpoint is capable of sending or receiving when this configuration is selected. Used here to pass 1-byte status information. Set to 1.
6	bInterval	1	Number	Left to the designer's discretion. A value of 10 ms or more seems sufficient.

4.4.3 Isochronous Audio Data Endpoint Descriptor

The standard and class-specific audio data endpoint descriptors provide pertinent information on how audio data streams are communicated to the audio function. Also, specific endpoint capabilities and properties are reported.

4.4.3.1 Standard Isochronous Audio Data Endpoint Descriptor

The standard isochronous audio data endpoint descriptor is identical to the standard endpoint descriptor defined in section 9.7.4 of the *USB Specification*. Its fields are set to reflect the isochronous type of the endpoint. Bit 7 of the **bEndpointAddress** field indicates whether the endpoint is an audio source (bit7 = 1) or an audio sink (bit7 = 0).

Table 4-5: Standard Isochronous Audio Data Endpoint Descriptor

Offset	Field	Size	Value	Description
0	bLength	1	Number	Size of this descriptor in bytes: 7
1	bDescriptorType	1	Constant	ENDPOINT descriptor type
2	bEndpointAddress	1	Endpoint	The address of the endpoint on the USB device described by this descriptor. The address is encoded as follows: Bit 0..3: The endpoint number, determined by the designer. Bit 4..6: Reserved, reset to zero. Bit 7: Direction: 0 = OUT endpoint for sinks. 1 = IN endpoint for sources.
3	bmAttributes	1	Bit Map	Bit 0..1: Transfer Type 01 = Isochronous All other bits are reserved.
4	wMaxPacketSize	2	Number	Maximum packet size this endpoint is capable of sending or receiving when this configuration is selected. This is determined by the audio bandwidth constraints of the endpoint, the number of interleaved data streams in the cluster and the audio data format of this endpoint.
6	bInterval	1	Number	Interval for polling endpoint for data transfers, expressed in milliseconds. Must be set to 1.

4.4.3.2 Class-Specific Isochronous Audio Data Endpoint Descriptor

The synchronization type of the endpoint (as defined in section 5.10.4.1 of the *USB Specification*) is reflected in the **bSyncType** field of this descriptor. For asynchronous sinks and adaptive sources, the **bEndpointLink** field provides the link to the associated sync endpoint.

The descriptor provides a **bSysExCode** field that can be used to indicate vendor-specific extended capabilities for the endpoint. Further information can be provided through the use of vendor-specific endpoint descriptors.

Also, the sampling frequency capabilities of the endpoint are reported. Depending on the **bSamFreqType** field, the length of the descriptor varies and the interpretation of the trailing fields differs. Sampling frequencies occupy three bytes and are expressed in Hz in order to support oversampled, reduced bit-resolution systems (the range is from 0 to 16,777,216 Hz).

Table 4-6: Class-Specific Isochronous Audio Data Endpoint Descriptor

Offset	Field	Size	Value	Description
0	bLength	1	Number	Size of this descriptor in bytes: 8 + n_s * 3
1	bDescriptorType	1	Constant	ENDPOINT descriptor type
2	bSyncType	1	Number	Synchronization type: 1 Asynchronous 2 Synchronous 3 Adaptive All other values are reserved.
3	bEndpointLink	1	Endpoint	The address of the endpoint that is used for synchronization in association with this endpoint. The address is encoded as follows: Bit 0..3: The endpoint number of the associated sync endpoint. Bit 4..6: Reserved, reset to zero. Bit 7: Direction: 0 = OUT endpoint for adaptive sources. 1 = IN endpoint for asynchronous sinks. Only used for adaptive sources and asynchronous sinks. Ignored in all other cases.
4	bmAttributes	1	Bit Map	A bit set to 1 indicates that the mentioned property is supported by this endpoint. Bit 0: Pitch control property Bit 7: System-exclusive properties All other bits are reserved.
5	bSysExCode	1	Number	Vendor-specific code indicating extended endpoint capabilities not covered by this document.
6	bDelay	1	Number	Delay (δ) introduced by this endpoint (see section 3.6). Expressed in number of frames.
7	bSamFreqType	1	Number	Indicates how the sampling frequency can be programmed. 0: Fixed sampling frequency 1: Continuous sampling frequency 2..255: n_s The number of discrete sample frequencies supported by the endpoint.
8...				See sampling frequency tables, below.

Depending on the value in the **bSamFreqType** field, the layout of the next part of the descriptor is as shown in the following tables.

Table 4-7: Fixed Sampling Frequency

Offset	Field	Size	Value	Description
8	wSamFreq	3	Number	Sampling frequency in Hz for this endpoint.

Table 4-8: Continuous Sampling Frequency

Offset	Field	Size	Value	Description
8	wLowerSamFreq	3	Number	Lower bound in Hz of the sampling frequency range for this endpoint.
11	wUpperSamFreq	3	Number	Upper bound in Hz of the sampling frequency range for this endpoint.

Table 4-9: Discrete Number of Sampling Frequencies

Offset	Field	Size	Value	Description
8	wSamFreq [1]	3	Number	Sampling frequency 1 in Hz for this endpoint.
11	wSamFreq [2]	3	Number	Sampling frequency 2 in Hz for this endpoint.
	…	…	…	…
$8+n_s*3$	wSamFreq [n_s]	3	Number	Sampling frequency n_s in Hz for this endpoint.

4.4.4 Isochronous Sync Endpoint Descriptor

This descriptor is present only when an associated isochronous audio data endpoint of the adaptive source type or the asynchronous sink type is implemented.

4.4.4.1 Standard Isochronous Sync Endpoint Descriptor

The isochronous sync endpoint descriptor is identical to the standard endpoint descriptor defined in section 9.7.4 of the *USB Specification*. Its fields are set to reflect the isochronous type of the endpoint.

Table 4-10: Standard Isochronous Sync Endpoint Descriptor

Offset	Field	Size	Value	Description
0	bLength	1	Number	Size of this descriptor in bytes: 7
1	bDescriptorType	1	Constant	ENDPOINT descriptor type
2	bEndpointAddress	1	Endpoint	The address of the endpoint on the USB device described by this descriptor. The address is encoded as follows: Bit 0..3: The endpoint number, determined by the designer. Bit 4..6: Reserved, reset to zero. Bit 7: Direction: 0 = OUT endpoint for sources. 1 = IN endpoint for sinks.
3	bmAttributes	1	Bit Map	Bit 0..1: Transfer type: 01 = Isochronous All other bits are reserved.
4	wMaxPacketSize	2	Number	Maximum packet size this endpoint is capable of sending or receiving when this configuration is selected. Endpoint is used to transfer 4-byte F_f information. Set to 4.
6	bInterval	1	Number	Interval for polling endpoint for data transfers, expressed in milliseconds. Must be set to 1.

4.4.4.2 Class-Specific Isochronous Sync Endpoint Descriptor

The class specific isochronous sync endpoint descriptor provides the link to the associated isochronous audio data endpoint. It further contains the **bUpdateCode** field that can be used by the driver to calculate the update interval for F_f as described in section 3.7.1.2.

Table 4-11: Class-Specific Isochronous Sync Endpoint Descriptor

Offset	Field	Size	Value	Description
0	bLength	1	Number	Size of this descriptor in bytes: 3
1	bDescriptorType	1	Constant	ENDPOINT descriptor type
2	bEndpointLink	1	Endpoint	The address of the isochronous audio data endpoint that is used in association with this sync endpoint. The address is encoded as follows: Bit 0..3: The endpoint number. Bit 4..6: Reserved, reset to zero Bit 7: Direction: 1 = IN endpoint for adaptive sources. 0 = OUT endpoint for asynchronous sinks.
3	bUpdateCode	1	Number	P value as implemented by the designer.

5. Requests

5.1 Standard Requests

The Audio Device Class supports the standard requests described in Chapter 9, “USB Device Framework,” of the *USB Specification*. The Audio Device Class places no specific requirements on the values for the standard requests.

5.2 Class-Specific Requests

Most class-specific requests get or set audio properties. These properties fall into two main groups: those that control audio functions, such as volume and tone, and those that control data transfer over the endpoint, such as sampling frequency.

- **Audio function control properties**. Control of an audio function is performed through the manipulation of audio properties of the audio control blocks (ACBs). The class-specific interface descriptor indicates which general control properties are supported for each ACB. Each ACB contains a field that can be used to extend its capabilities with vendor-specific control properties. All control property requests must be directed to the audio interface.
- **Endpoint properties**. Control of the class-specific behavior of the endpoint is performed through manipulation of endpoint properties. All endpoint property requests must be directed to the isochronous audio data endpoint.

Within these two groups, properties may be general or system-exclusive:

- **General properties**. Properties that should be supported by all USB Audio devices. The **Get/Set Property** requests are used to get or set general audio properties. General properties are described in section 5.3.
- **System-exclusive properties**. A vendor may define additional properties, called extended or system-exclusive properties. The **Get/Set SysEx Property** requests are used to get or set system-exclusive properties. For information about system-exclusive properties, see the vendor documentation.

The Audio Device Class supports two additional class-specific requests:

- The **Get Error** request can be used to get error information for either general or system-exclusive properties.
- The **Get Status** request is a general query to the interface and does not manipulate properties.

The rest of this section describes the class-specific requests used to manipulate both general and system-exclusive properties, followed by a description of the general properties supported by the Audio Device Class.

5.2.1 Set General Property Request

This request is used to set a general property of a particular ACB or endpoint.

Table 5-1: Set General Property Request Values

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00100001B	SET_GEN_PROP	ACB index and control property selector	Interface	Length of property parameter(s)	Property parameter(s)
00100010B		Endpoint property selector	Endpoint		

If the request is sent to an interface, the **wValue** field specifies the index of the audio control block in the high byte and the general control property selector in the low byte. If the request is sent to an endpoint, the **wValue** field specifies 0 in the high byte and the general endpoint property selector in the low byte. The values in **wValue** must be appropriate to the recipient. Only existing ACBs in the audio function can be addressed and only general property selector values may be used. If the request specifies an unknown ACB index or selector, the control pipe must indicate a stall.

The **wIndex** field specifies the interface or endpoint to be addressed.

The actual parameter(s) for the property are passed in the data stage of the control transfer. The length of the parameter block is indicated in the **wLength** field of the request. For a description of the parameter block for each general control property, see section 5.3.1. For a description of the parameter block for each general endpoint property, see section 5.3.

5.2.2 Get General Property Request

This request returns the current setting of a specific general property.

Table 5-2: Get General Property Request Values

bmRequestType	bRequest	wValue	wIndex	wLength	Data
10100001B	GET_GEN_PROP	ACB index and control property selector	Interface	Length allocated for the returned property setting.	Current property setting
10100010B		Endpoint property selector	Endpoint		

If the request is sent to an interface, the **wValue** field specifies the index of the audio control block in the high byte and the general control property selector in the low byte. If the request is sent to an endpoint, the **wValue** field specifies 0 in the high byte and the general endpoint property selector in the low byte. The values in **wValue** must be appropriate to the recipient. Only existing ACBs in the audio function can be addressed and only general control or endpoint property selector values may be used. If the request specifies an unknown ACB index or selector, the control pipe must indicate a stall.

The **wIndex** field specifies the interface or endpoint to be addressed.

The actual parameter(s) for the property are returned in the data stage of the control transfer. The length of the parameter block to return is indicated in the **wLength** field of the request. If the property setting is longer than the **wLength** field, only the initial bytes of the property setting are returned. If the property setting is shorter than the **wLength** field, the function indicates the end of the control transfer by sending a short packet when further data is requested. For a description of the parameter block for each general audio control property, see section 5.3.1. For a description of the parameter block for each general endpoint property, see section 5.3.2

5.2.3 Set SysEx Property Request

This request is used to set a system-exclusive property of a particular ACB or endpoint.

Table 5-3: Set SysEx Property Request Values

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00100001B	SET_SYSEX_PROP	ACB index and system-exclusive code	Interface	Length of the parameter block length	Parameter block
00100010B		System-exclusive code	Endpoint		

If the request is sent to an interface, the **wValue** field specifies the index of the audio control block in the high byte and the system-exclusive code in the low byte. If the request is sent to an endpoint, the **wValue** field specifies 0 in the high byte and the system-exclusive code in the low byte. The values in **wValue** must be appropriate to the recipient. Only existing ACBs in the audio function can be addressed, and only vendor-specific system-exclusive code values may be used. If the request specifies an unknown ACB index or system-exclusive code, the control pipe must indicate a stall.

The actual parameter block for the system exclusive code is passed in the data stage of the control transfer. The length of the parameter block is indicated in the **wLength** field of the request. The contents of the parameter block are vendor-specific and is not covered by this document.

5.2.4 Get SysEx Property Request

This request returns the current setting of a system-exclusive property.

Table 5-4: Get SysEx Property Request Values

bmRequestType	bRequest	wValue	wIndex	wLength	Data
10100001B	GET_SYSEX_PROP	ACB index and system-exclusive code	Interface	Length allocated for the returned property setting.	Current parameter block
10100010B		System-exclusive code	Endpoint		

The **wValue** field specifies the index of the audio control block in the high byte and the system-exclusive code in the low byte. The values in **wValue** must be appropriate to the recipient. Only existing ACBs in the audio function can be addressed, and only vendor-specific system-exclusive code values may be used. If the request specifies an unknown ACB index or system-exclusive code, the control pipe must indicate a stall.

The actual parameter block for the system-exclusive code is returned in the data stage of the control transfer. The length of the parameter block to return is indicated in the **wLength** field of the request. If the parameter block is longer than the **wLength** field, only the initial bytes of the parameter block are returned. If the parameter block is shorter than the **wLength** field, the function indicates the end of the control transfer by sending short packet when further data is requested. The contents of the returned parameter block are vendor-specific and is not covered by this document.

5.2.5 Get Error Request

This request is used to retrieve error messages from the audio function. The occurrence of an error is reported to the host through the **sError** bit in the Status Byte over the status interrupt endpoint. This bit will remain set until all currently available error messages have been read.

Table 5-5: Get Error Request Values

bmRequestType	bRequest	wValue	wIndex	wLength	Data
10100001B	GET_ERROR	Zero	Interface	Error message length	Error message

The **wLength** field specifies the number of bytes to return. If the error message is longer than the **wLength** field, only the initial bytes of the error message are returned. If the error message is shorter than the **wLength** field, the function indicates the end of the control transfer by sending short packet when further data is requested.

Error messages for general audio properties are covered by this document and follow a predefined format. The message is always 5 bytes long according to the following table.

Table 5-6: Error Message Format for General Audio Properties

Offset	Field	Size	Value	Description
0	bLength	1	Number	Size of the error message in bytes: 5
1	bGenPropError	1	Number	GENERAL_CONTROL_PROPERTY or GENERAL_ENDPOINT_PROPERTY
2	bErrorSource	1	Number	Index of the ACB or endpoint address that caused the last error.
3	bGenProp	1	Selector	Property selector of the property that caused the last error.
4	bErrorCode	1	Number	The property-specific error code.

Error messages for system-exclusive properties are not covered by this document and follow a free format. The message has a variable length and is built up according to the following table.

Table 5-7: Error Message Format for System-Exclusive Properties

Offset	Field	Size	Value	Description
0	bLength	1	Number	Size of the error message in bytes: 4+n
1	bSysExError	1	Constant	SYSTEM_EXCLUSIVE
2	bErrorSource	1	Number	Index of the ACB or endpoint address that caused the last error.
3	bSysExCode	1	Number	System-exclusive code of the property that caused the last error.
4	bErrorCode	n	Number	The system-exclusive error message.

5.2.6 Get Status Request

This request is used to retrieve status information from the audio function.

Table 5-8: Get Status Request Values

bmRequestType	bRequest	wValue	wIndex	wLength	Data
10100001B	GET_STATUS	Zero	Interface	Status message length	Status message

The **wLength** field specifies the number of bytes to return. If the status message is longer than the **wLength** field, only the initial bytes of the status message are returned. If the status message is shorter than the **wLength** field, the function indicates the end of the control transfer by sending short packet when further data is requested.

The contents of the status message are TBD.

The appropriate bits in the status byte remain set until all available status messages have been read.

5.3 General Audio Properties

5.3.1 General Audio Control Properties

The following sections describe the possible general control properties an audio function can support through its ACBs. The layout of the parameter blocks is used for **Set/Get Property** requests.

5.3.1.1 Mute Property

The **Mute** property controls the position of the mute switch in an ACB. The position can be either on or off.

Table 5-9: Mute Property Parameter Block

Property selector		MUTE_SELECTOR		
Parameter Block Length		1		
Parameter Block				
Offset	**Field**	**Size**	**Value**	**Description**
0	bOn	1	Bool	ACB muted when TRUE, not muted when FALSE.
Error Codes				
0x00	No error			
0x01	Property not supported			

5.3.1.2 Volume Property

The **Volume** property controls the volume of the ACB. The range is from 0 dB down to -∞ dB in steps of 1/256 dB (0.00390625 dB), except for the last step which goes from -255.9922 dB to -∞ dB. The ACB honors the request to the best of its abilities. It may round the **bVolume** value to its closest available setting. It will report this setting when queried during a **Get Property** request. The setting 0 dB is always referenced to the maximum available volume setting.

Table 5-10: Volume Property Parameter Block

Property selector	VOLUME_SELECTOR			
Parameter Block Length	1			
Parameter Block				
Offset	**Field**	**Size**	**Value**	**Description**
0	bVolume	2	Number	The volume setting: 0x0000: 0.0 dB 0x0001: -0.0039 dB 0x0002: -0.0078 dB … 0x0100: -1.0000 dB … 0xFFFD: -255.9922 dB 0xFFFE: -255.9961 dB 0xFFFF: -∞ dB

5.3.1.3 Bass, Mid, and Treble Properties

The tone control properties **Bass**, **Mid**, and **Treble** influence the general tone control behavior of the ACB. The range is from +31.75 dB down to –31.75 dB in steps of 0.25 dB. The ACB honors the request to the best of its abilities. It may round the **bBass|bMid|bTreble** value to its closest available setting. It will report this setting when queried during a **Get Property** request. Other parameters that also influence the behavior of the tone control settings, such as cut-off frequency, cannot be altered through this request and are left to the designer. If desired, they can be defined as system-exclusive properties that can be accessed using the **Get/Set SysEx Property** request.

Table 5-11: Bass, Mid, and Treble Property Parameter Block

Property selector	BASS_SELECTOR, MID_SELECTOR, TREBLE_SELECTOR			
Parameter Block Length	1			
Parameter Block				
Offset	**Field**	**Size**	**Value**	**Description**
0	bBass bMid bTreble	1	Number	The tone control setting. 0x7F: +31.75 dB 0x7E: +31.50 dB … 0x00: 0.00 dB … 0x82: -31.50 dB 0x81: -31.75 dB 0x80: -32.00 dB

5.3.1.4 Graphic Equalizer Property

The Audio Device Class definition provides for standard support of a third octave graphic equalizer. The bands are defined according to the ANSI S1.11-1986 standard. Bands are numbered from 14 (center frequency of 25 Hz) up to 43 (center frequency of 20,000 Hz), making a total of 30 possible bands. An ACB that supports the graphic equalizer property is not required to implement the full set of filters. A subset (for example, octave bands) may be implemented. The first field in the parameter block is a bitmap indicating which bands are effectively implemented. The number of bits set in this field determine the total length of the parameter block.

Table 5-12: Graphic Equalizer Property Parameter Block

Property selector		GRAPHIC_EQUALIZER_SELECTOR		
Parameter Block Length		30		
Parameter Block				
Offset	**Field**	**Size**	**Value**	**Description**
0	bmBandsPresent	4	Bit Map	A bit set indicates the band is present: Bit 0: Band 14 is present Bit 1: Band 15 is present … Bit 29: Band 43 is present Bit 30: Reserved Bit 31: Reserved
5	bBand14	1	Number	The gain setting for the 25Hz band. 0x7F: +31.75 dB 0x7E: +31.50 dB … 0x00: 0.00 dB … 0x82: -31.50 dB 0x81: -31.75 dB 0x80: -32.00 dB
1	bBand15 *	1	Number	The gain setting for the 31.5Hz band.
2	bBand16	1	Number	The gain setting for the 40Hz band.
3	bBand17	1	Number	The gain setting for the 50Hz band.
4	bBand18 *	1	Number	The gain setting for the 63Hz band.
5	bBand19	1	Number	The gain setting for the 80Hz band.
6	bBand10	1	Number	The gain setting for the 100Hz band.
7	bBand21 *	1	Number	The gain setting for the 125Hz band.
8	bBand22	1	Number	The gain setting for the 160Hz band.

Property selector	GRAPHIC_EQUALIZER_SELECTOR			
Parameter Block Length	30			
Parameter Block				
Offset	**Field**	**Size**	**Value**	**Description**
9	bBand23	1	Number	The gain setting for the 200Hz band.
10	bBand24 *	1	Number	The gain setting for the 250Hz band.
11	bBand25	1	Number	The gain setting for the 315Hz band.
12	bBand26	1	Number	The gain setting for the 400Hz band.
13	bBand27 *	1	Number	The gain setting for the 500Hz band.
14	bBand28	1	Number	The gain setting for the 630Hz band.
15	bBand29	1	Number	The gain setting for the 800Hz band.
16	bBand30 *	1	Number	The gain setting for the 1000Hz band.
17	bBand31	1	Number	The gain setting for the 1250Hz band.
18	bBand32	1	Number	The gain setting for the 1600Hz band.
19	bBand33 *	1	Number	The gain setting for the 2000Hz band.
20	bBand34	1	Number	The gain setting for the 2500Hz band.
21	bBand35	1	Number	The gain setting for the 3150Hz band.
22	bBand36 *	1	Number	The gain setting for the 4000Hz band.
23	bBand37	1	Number	The gain setting for the 5000Hz band.
24	bBand38	1	Number	The gain setting for the 6300Hz band.
25	bBand39 *	1	Number	The gain setting for the 8000Hz band.
26	bBand40	1	Number	The gain setting for the 10000Hz band.
27	bBand41	1	Number	The gain setting for the 12500Hz band.
28	bBand42 *	1	Number	The gain setting for the 16000Hz band.
29	bBand43	1	Number	The gain setting for the 20000Hz band.

Note: Bands marked with an asterisk (*) are those present in an octave equalizer.

5.3.1.5 Emphasis Property

The **Emphasis** property controls the position of the emphasis switch in an ACB. The position can be either on or off.

Table 5-13: Emphasis Property Parameter Block

Property selector		EMPHASIS_SELECTOR		
Parameter Block Length		1		
Parameter Block				
Offset	**Field**	**Size**	**Value**	**Description**
0	bOn	1	Bool	Emphasis on when TRUE, off when FALSE.

5.3.2 General Endpoint Properties

The following sections describe the properties that an audio function can support through its endpoints. The layout of the parameter blocks is used for **Set/Get Property** requests.

5.3.2.1 Sampling Frequency Property

The **Sampling Frequency** property is used to set the initial sampling frequency for an isochronous audio data endpoint. If the endpoint operates at a fixed sampling frequency, setting this property has no effect. If the endpoint supports a discrete number of sampling frequencies, setting the sampling frequency to a non-existent value causes the control pipe to indicate a stall. This also happens when the sampling frequency is set outside the range for a continuous sampling frequency endpoint.

Table 5-14: Sampling Frequency Property Parameter Block

Property selector		SAMPLING_FREQUENCY_SELECTOR		
Parameter Block Length		3		
Parameter Block				
Offset	**Field**	**Size**	**Value**	**Description**
0	bSamFreq	3	Number	The sampling frequency, expressed in Hz.

5.3.2.2 Pitch Control Property

The **Pitch Control** property enables or disables the ability of an adaptive endpoint to dynamically track its sampling frequency. The property is necessary because the clock recovery circuitry must be informed whether it should allow for relatively large swings in the sampling frequency. The pitch control can be either on or off.

Table 5-15: Pitch Control Property Parameter Block

Property selector		PITCH_ SELECTOR		
Parameter Block Length		1		
Parameter Block				
Offset	**Field**	**Size**	**Value**	**Description**
0	bOn	1	Bool	Pitch control on when TRUE, off when FALSE.

6. Subclasses and Protocols

6.1 8-bit PCM Mono Audio Data

Subclass	SC_PCM_DATA_8_BIT	
Protocol	PR_MONO	
Interface	1 control endpoint 1 status interrupt endpoint 1 isochronous audio data endpoint	
	Format	8 bits mono PCM audio samples. MSb first offset binary.

6.2 8-bit PCM Stereo Audio Data

Subclass	SC_PCM_DATA_8_BIT	
Protocol	PR_STEREO	
Interface	1 control endpoint 1 status interrupt endpoint 1 isochronous audio data endpoint	
	Format	8 bits stereo PCM audio samples. MSb first offset binary.

6.3 16-bit PCM Mono Audio Data

Subclass	SC_PCM_DATA_16_BIT	
Protocol	PR_MONO	
Interface	1 control endpoint 1 status interrupt endpoint 1 isochronous audio data endpoint	
	Format	16-bits mono PCM audio samples. MSb first two’s complement.

6.4 16-bit PCM Stereo Audio Data

Subclass	SC_PCM_DATA_16_BIT	
Protocol	PR_STEREO	
Interface	1 control endpoint 1 status interrupt endpoint 1 isochronous audio data endpoint	
	Format	16 bits stereo interleaved PCM audio samples. MSb first two's complement.

6.5 16-bit PCM Quadro Audio Data

Subclass	SC_PCM_DATA_16_BIT	
Protocol	PR_QUADRO	
Interface	1 control endpoint 1 status interrupt endpoint 1 isochronous audio data endpoint	
	Format	16 bits four channel interleaved PCM audio samples. MSb first two's complement.

6.6 16-bit PCM Stereo&Stereo Audio Data

Subclass	SC_PCM_DATA_16_BIT	
Protocol	PR_STEREO&STEREO	
Interface	1 control endpoint 1 status interrupt endpoint 2 isochronous audio data endpoints	
	Format	16 bits stereo interleaved PCM audio samples. MSb first two's complement.

6.7 16-bit Dolby Surround™ Encoded Data

Subclass	SC_DOLBY_SURROUND_DATA_16_BIT	
Protocol	PR_STEREO	
Interface	1 control endpoint 1 status interrupt endpoint 1 isochronous audio data endpoint	
	Format	16 bits stereo interleaved audio samples. MSb first two's complement. Dolby Surround™ encoded.

6.8 Stereo Audio Encoded According to IEC958 - Consumer Mode

Subclass	SC_IEC958	
Protocol	PR_IEC958_CONSUMER	
Interface	1 control endpoint 1 status interrupt endpoint 1 isochronous audio data endpoint	
	Format	32 bit stereo interleaved audio samples with auxiliary information. The 32 bits are used to encode a synchronous 20- or 24-bit stereo audio channel with additional bits, as described in the international IEC-958 Digital Audio Interface standard. Bit 0..3 of each 32-bit 'extended sample' (sub-frame) contain the sync preamble. Bit 4..27 contains the PCM coded audio sample word. Bit 28 is the Validity Flag, bit 29 contains the User Data, bit 30 contains the Channel Status. Bit 31 is the Parity bit. An in-depth explanation of C, U, and V bits is beyond the scope of this document. Refer to the IEC-958 standard for details.

6.9 Stereo Audio Encoded According to IEC958 - Professional Mode

Subclass	SC_IEC958	
Protocol	PR_IEC958_PROFESSIONAL	
Interface	1 control endpoint 1 status interrupt endpoint 1 isochronous audio data endpoint	
	Format	32 bit stereo interleaved audio samples with auxiliary information. The 32 bits are used to encode a synchronous 20- or 24-bit stereo audio channel with additional bits, as described in the international IEC-958 Digital Audio Interface standard. Bit 0..3 of each 32-bit 'extended sample' (sub-frame) contain the sync preamble. Bit 4..27 contains the PCM coded audio sample word. Bit 28 is the Validity Flag, bit 29 contains the User Data, bit 30 contains the Channel Status. Bit 31 is the Parity bit. An in-depth explanation of C, U, and V bits is beyond the scope of this document. Refer to the IEC-958 standard for details.

6.10 Stereo Audio Encoded According to MPEG1 - Layer 1

Subclass	SC_MPEG1	
Protocol	PR_LAYER1	
Interface	TBD	
	Format	TBD

6.11 Stereo Audio Encoded According to MPEG1 - Layer 2

Subclass	SC_MPEG1	
Protocol	PR_LAYER2	
Interface	TBD	
	Format	TBD

6.12 Stereo Audio Encoded According to AC3

Subclass	SC_AC3	
Protocol	TBD	
Interface	TBD	
	Format	TBD

Appendix A Audio Device Class Codes

A.1 Audio Interface Class Code

Audio Interface Class Code	Value
C_AUDIO	0x01

A.2 Audio Subclass Codes

Subclass Code	Value
SC_PCM_DATA_8_BIT	0x01
SC_PCM_DATA_16_BIT	0x02
SC_DOLBY_SURROUND_DATA_16_BIT	0x03
SC_IEC958	0x04
SC_MPEG1	0x05
SC_AC3	0x06

A.3 Audio Protocol Codes

Protocol Code	Value
PR_MONO	0x01
PR_STEREO	0x02
PR_QUADRO	0x03
PR_STEREO&STEREO	0x04
PR_IEC958_CONSUMER	0x05
PR_IEC958_PROFESSIONAL	0x06
PR_LAYER1	0x07
PR_LAYER2	0x08

A.4 Class-Specific Request Codes

Class-Specific Request Code	Value
SET_GENERAL_PROPERTY	0x01
GET_GENERAL_PROPERTY	0x02
SET_SYSEX_PROPERTY	0x03
GET_SYSEX_PROPERTY	0x04
GET_ERROR	0x05
GET_STATUS	0x06

A.5 General Property Selector Codes

A.5.1 Audio Control Property Selectors

General Control Property Selector	Value
MUTE_SELECTOR	0x01
VOLUME_SELECTOR	0x02
BASS_SELECTOR	0x03
MID_SELECTOR	0x04
TREBLE_SELECTOR	0x05
GRAPHIC_EQUALIZER_SELECTOR	0x06
EMPHASIS_SELECTOR	0x07

A.5.2 Endpoint Property Selectors

General Control Property Selector	Value
SAMPLING_FREQUENCY_SELECTOR	0x01
PITCH_SELECTOR	0x02

A.6 Error Codes

TBD